A Study of the Extension Development Model of WebKit

Xiao Ma
School of Information Engineering
Communication University of China
Beijing, China

Liming Ma
School of Information Engineering
Communication University of China
Beijing, China

Sanxing Cao
New Media Institute
Communication University of China
Beijing, China

***Abstract*—This paper, integrating the practical development demand on the B/S architecture application, presents an extension development model of WebKit Browser Engine. The composition, compilation process and the extension development principles of WebKit are illustrated. Taking account of the different requirements of browser engine extension development in the actual projects, we propose the strategies for the extension development of WebKit including source code modification for extension and API-based extension. Further more, we refine the strategy of API-based extension by introducing three specific ways as rewriting the existing API, developing plugins and extending JavaScript Object. The strategies have achieved a satisfied performance by applying in the actual project.**

Keywords-WebKit; Browser Engine; Extension; Plugin

I. INTRODUCTION

B/S based software implementation is becoming the mainstream technology approach in application development, owing to its low cost of deployment, agile iterative upgrading, cross-platform support, and other advantages. The functional logic can run on the various platforms by writing HTML, CSS and JavaScript code in once. The browser, playing the role of "Uniform Client", is responsible for the internal underlying work of network communications, content rendering and script interpreting. It provides a user-interface to present the page and responds to the user's interactions. The basic composition of the browser is shown in Fig. 1. The Engine is the core component of a browser. WebKit, Gecko and Trident are among the most widely used browser engines at present.

In the process of software project implementation based on Browser/Server (hereinafter referred as B/S) architecture, the main approach for traditional Web application development, which highly relies on the server-side programs in service logic implementation, is to do the coding work aiming at the Generic Browser directly. But the advanced Web application always requires the adequate access to the local computing capabilities and resources to meet specific application demands, which leads to the strategies for the extension development of WebKit including source code modification for extension and API-based extension. It will provide strong support for the development and operation of Web application by studying the WebKit browser engine.

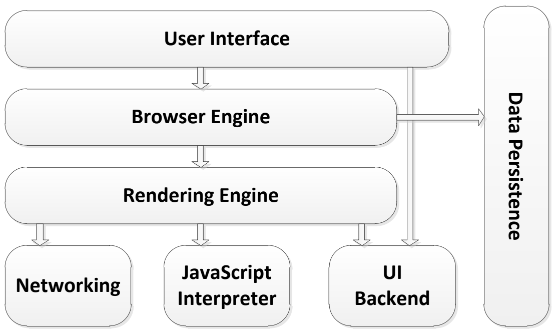


Figure 1. The Basic Composition of Browser

II. THE BASIS OF WEBKIT

A. Key Features of WebKit

WebKit is an open source browser engine with the features of clear, stable, efficient, maintainable and compatible for the Web standards. It is adopted in Apple Safari, Google Chrome and other modern browsers. Apple WebKit, QtWebKit and Chromium are the main branches of the WebKit Open Source Project. Apple WebKit contains the primitive source code of the project. Its code size is less than Chromium, while the building environment is relatively harsh. Chromium is unique because it employs the V8 Engine as the JavaScriptCore. As WebKit is packaged and integrated in QtWebKit, the development based on QtWebKit is relatively convenient, since it is not complicated to transplant due to the cross-platform characteristic of Qt.

B. The Composition of WebKit

In the sight of source code hierarchical structure, WebKit, WebCore and JavaScriptCore are involved in the WebKit Engine.

- WebKit: It is is at the top of the source code package, and defines a series of interfaces for interacting with the application, providing a corresponding implementation for different platforms according to the requirement of cross-platform applications.

- WebCore: It is is the core component of WebKit, which contains the Layout Engine for HTML document and SVG, Render Engine, DOM Library and other components.
- JavaScriptCore: It is is a framework that contains the Script Interpreter, Analyzer and Execution Procedure implementing a JavaScript Engine for WebKit. It supports the JavaScript language by explaining and executing the program. Its running process includes syntax analysis, syntax tree construction, variable checking, and syntax tree execution.

Fig. 2 shows a basic architecture of WebKit. It describes the relationship between WebKit, WebCore, JavaScriptCore and the libraries that they rely on.

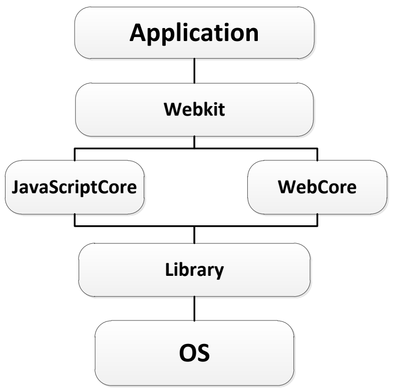


Figure 2. Basic Architecture of WebKit

C. The Compilation of WebKit

By comparing the main branches of WebKit, we carry on the extension development based on QtWebKit. The source code of QtWebKit can be downloaded via the Internet directly. It can be compiled by means of Perl script parameters.

1) Get Source Code

The source code of QtWebKit is hosted on the gitorious site by the method of Git, which can be gotten by git command:

```
git clone git://gitorious.org/+qtwebkit-developers/webkit/qtwebkit.git
```

2) Compilation Process

a) Command Line Mode Compilation

It is easy to compile QtWebKit by using the Perl compile script "build-webkit" with "-qt" compile option. Other compile options are available by typing "--help" parameter to access.

b) Generate the Project Files of Visual Studio

It demands Perl script to generate a lot of source codes before the compilation of WebKit. The compilation can't be passed as it lacks the necessary header files and source files if we use the "qmake" command for converting the ".pro" project file to the ".sln" solution files for Visual Studio directly. So it is necessary to generate the source files that compilation required before generating the ".sln" file.

The detailed command for generation is as follows:

```
qmake -r ../Source/DerivedSources.pro –o Makefile.DerivedSources
nmake -f Makefile.DerivedSources generated_files
qmake -r -tp vc ../Source/webkit.pro
```

In addition, there are several compile options can be added when executing the "qmake" command. For example, it will support the video tag in QtWebKit when attaching the "DEFINES + = ENABLE_VIDEO = 1" compile option.

III. SOURCE CODE MODIFICATION FOR EXTENSION

As WebKit is an open source Browser Engine, theoretically, the source code of it can be modified flexibly to achieve the purpose of extension development by accessing to the source code. The example of extending window.open function in JavaScript shown below introduces the extension method of source code modification.

Firstly, it is necessary to trace the invoked process of window.open function in WebKit to find the involved source code.

WebKit will invoke the createWindow function located in JSDOMWindowCustom.cpp that is the helper function of window.open belonging to the WebCore namespace when JavaScript Interpreter executes the window.open function. Then, the function will invoke the overloaded function createWindow also in the WebCore namespace, which is in charge of the remaining logics. Fig. 3 shows the process of invoking and the corresponding source files.

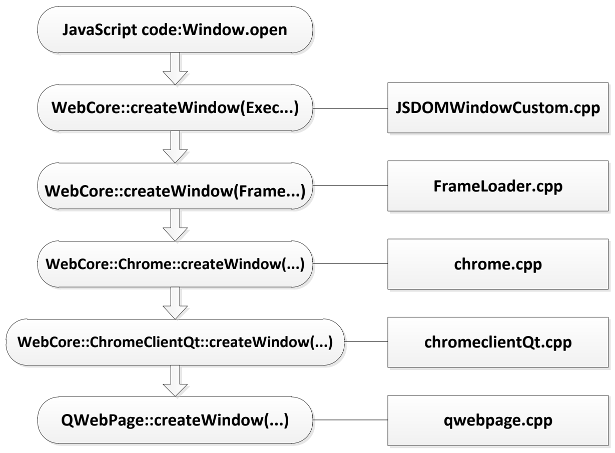

JavaScript code:Window.open
WebCore::createWindow(Exec...)
JSDOMWindowCustom.cpp
WebCore::createWindow(Frame...)
FrameLoader.cpp
WebCore::Chrome::createWindow(...)
chrome.cpp
WebCore::ChromeClientQt::createWindow(...)
chromeclientQt.cpp
QWebPage::createWindow(...)
qwebpage.cpp

Figure 3. The Invoking Process of window.open in WebKit

```
Page* page = oldPage->chrome()->createWindow(openerFrame, requestWithReferrer, frameName, features, action);
```

After obtaining a pointer whose type is WebCore::Chrome by oldPage->chrome() method, it will invoke createWindow function that is the member function of Chrome class.

```
Page* newPage = m_client->createWindow(frame, request, frameName, features, action);
```

And then the m_client pointer invokes its member function createWindow. The pointer is WebCore::ChromeClientQt type.

```
QWebPage *newPage = m_webPage->createWindow(features.dialog ? QWebPage::WebModalDialog : QWebPage::WebBrowserWindow );
```

The createWindow virtual function, which is belonging to the QWebPage class, is the key function that is required to customize. Its prototype is as follows:

```
QWebPage *QWebPage::createWindow(WebWindowType type)
```

After understanding the invoking process, It is clear that the main parameter to control the feature of the pop-up window ultimately is a structure called WindowFeatures. There is a declaration for the structure in WindowFeatures.h. The Boolean type variables in the structure are the corresponding properties of the pop-up window. For example, "maxsize" controls to maximize the window.

Therefore, in order to set the style of pop-up window, it is demanded to pass the Boolean values of corresponding member variables in WindowFeatures to the member function createWindow of WebCore::ChromeClientQt. For additional style setting requirements, it is necessary to build a customized structure based on WindowFeatures whose name can be called MyWF, and overload the function QWebPage::createWindow in WebCore::ChromeClientQt::createWindow to modify the realization of the logic.

```
QWebPage *QWebPage::createWindow(MyWF features)
{
        //Completely Implemented by Subclass
        return NULL;
}
```

As the code shown above, the attributes of the pop-up window are controlled by the customized structure MyWF.

IV. API-BASED EXTENSTION

As a matter of fact, the large volume of the WebKit source code has led to a high cost of understanding and modification. The direct strategy of source code modification for extension is not feasible for all situations. According to the scalability theory of software engineering, this paper puts forward the API-based extension strategy under QtWebKit APIs, which can be refined by introducing three specific ways as rewriting the existing API, developing plugins and extending JavaScript Object.

A. Rewriting the Existing API

WebKit is packaged perfectly in QtWebKit containing almost all of the important components such as security permission, DOM element access, history management etc.

It will be realized in practice by rewriting the QtWebKit interfaces for extension development, by which JavaScript and the browser engine can interact with each other. For example, the 'alert' function in JavaScript can be extended by rewriting the virtual function javaScriptAlert() belonging to QwebPage class in QtWebKit.

```
void WebPage::javaScriptAlert(QWebFrame *frame, const QString &msg)
{
        //The Rewritten Logic
}
```

B. Developing plugins

Taking advantage of QWebPluginFactory class provided by QtWebKit APIs is the effective way to develop the customized plugins, by which the plugins written in C++ can be added in the browser dynamically to extend the capabilities of it. Moreover, it supports JavaScript to control the plugins by the realization of utilizing JavaScript to invoke the member function of plugins.

The two pure virtual functions of QWebPluginFactory class shown below are the keys to realize the plugin:

```
virtual QObject * create (
        const QString & mimeType,
        const QUrl & url,
        const QStringList & argumentNames,
        const QStringList & argumentValues
) const;
virtual QList<QWebPluginFactory::Plugin> plugins () const ;
```

The implementation logics and settings of the plugin's property are added in the function 'create()'. Simultaneously, plugin is loaded after its options are enabled in the constructor function of QWebView. WebPluginFactory class is the implement of the abstract class QWebPluginFactory.

```
    this->page()->settings()->setAttribute(QWebSettings::PluginsEnabled,
true);
    this->page()->setPluginFactory(new WebPluginFactory(this));
```

WebKit will invoke the create function when <object> tag appears in the HTML document. The plugin is created by the parameters passing into the function, returning a pointer to the browser engine that will render the plugin finally. The member function of plugins will be invoked when JavaScript accesses the global plugin or MIME types object.

C. Extending JavaScript Object

In the QWebFrame class, there is a member function addToJavaScriptWindowObject() which is provided by the API in QtWebKit, adding the extension object to the root node named window in Browser Object Model (BOM). The extension object is a bridge in the application for the interoperation between C++ modules and JavaScript. Fig. 4 is the schematic diagram of extending JavaScript Object.

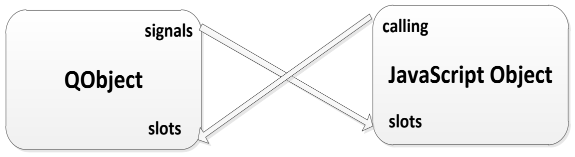


Figure 4. The Process of Extending JavaScript Object

The member function in the extended class written in C++ should be the type of "slots", as the document of Qt said, "Qt properties will be exposed as JavaScript properties and slots as JavaScript methods". Therefore, the member function in the extended class can interoperate with JavaScript.

```
//C++ Code

class MyHelper:public QObject
```

```
{
        public slots:void handler ();
};
```

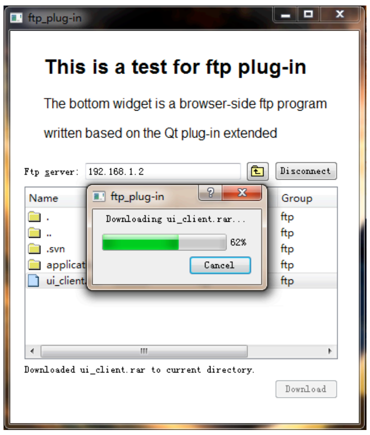
The extended class will be instantiated in the initialization process of browser engine. The instance of the extended class is mapped into a JavaScript Object under BOM by invoking the function addToJavaScriptWindowObject().

```
//C++ Code

FtpHelper *myhelper = new MyHelper ();
webframe->addToJavaScriptWindowObject("myhelper ", myhelper);
```

JavaScript can invoke the extension object after the process of initialization and mapping.

```
//JavaScript Code

myhelper.handler();
```

V. VERIFICATION AND CONCLUSION


This is a test for ftp plug-in

The bottom widget is a browser-side ftp program written based on the Qt plug-in extended

Ftp server: 192.168.1.2

Downloading ui_client.rar...

Downloaded ui_client.rar to current directory.

Figure 5. A Browser with FTP Plugin

The browser engine is the underlying technology to support the B/S architecture applications. In this paper, we focus on the open source browser engine WebKit, which is clear, stable, efficient, maintainable and compatible for the Web standards. Through the in-depth study of WebKit, we propose the strategies for the extension development of it including source code modification for extension and API-based extension, which demonstrates high operability and flexibility in actual software development projects. As an experimental demo, the browser with a FTP plugin is shown in Fig. 5, which adopts the strategies proposed above to add the FTP transmission capability to the browser.

In the follow-up work, we will further study the method to load the browser extension dynamically, trying to implement the modularization of different extensions and package the modules independently, decoupling the extensions with browser engine. Taking account of the specific needs of different Web applications, the extensions will be loaded from the external dynamically by certain mechanisms of distribution and deployment. It will simplify the difficulty for customized development of Web applications in different areas. It not only maintains the characteristics of low cost of deployment, agile iterative update, cross-platform feature, and other advantages on Web Application, but also can access to the local computing capabilities and resources adequately as Native Application.

We hereby hope that the approach of WebKit extension presented in this paper could act as an effective reference model for the information engineering applications of the kind.

ACKNOWLEDGMENT

This work is sponsored by the 382 Talents Plan (G08382304), Chinese National Hi-tech Industrialization Project(CNGI-09-03-15-1), and the Virtual Library Technology project granted by China Art Science and Technology Institute.

REFERENCES

[1] Yan Hongcan, Zhu Xiaoliang, Liu Xiaobin,and Wu Shangzhuo, “The Design and Realization of the Linux Browser Based on Webkit”, Test and Measurement, 2009. ICTM '09. International Conference on, vol. 2, pp. 188-191, 5-6 Dec. 2009

[2] Jiang Zhang-gai, Chen Rong, “Native extension strategy of WebKit based on CAR components”, JOURNAL OF COMPUTER APPLICATIONS, 2009, 29(z2)

[3] Stepan B. Sokolov, “Method and apparatus for interfacing a javascript interpreter with library of host objects implemented in java”, Nov 23, 2004

[4] Zhang Junlin, Yang Fumin, Hu Guanrong, “The Design and Implementation of a Javascript Interpreter”, Computer Engineering and Applications, vol.30, 2004

[5] Wu Guang-xu, Wu Xiao-xi, “Browser Based on WebKit Native Extension Method”, Computer Knowledge and Technolog， vol.30, 2011

[6] Saida, Y. ; Chishima, H. ; Hieda, S. ; Sato, N. ; Nakamoto, Y. ; NEC Network Dev. Labs, Yokohama, Japan, “An extensible browser architecture for mobile terminals”, Distributed Computing Systems Workshops, 2004. Proceedings. 24th International Conference on, 23-24 March 2004

[7] Taivalsaari, A. ; Mikkonen, T. ; Ingalls, D. ; Palacz, K. Sun Microsyst. Labs., Tampere, Finland, “Web Browser as an Application Platform”, Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference, 3-5 Sept. 2008, p293 – 302

[8] The WebKit Open Projects. http://webkit.org/

[9] The Qt Official Documentation, “Qt Signals and Slots”, http://developer.qt.nokia.com/doc/qt-4.8/signalsandslots.html